Learn to Build an End-to-End Machine Learning Pipeline - Part 3 Project Overview

Overview

The project addresses a critical challenge faced by the logistics industry. Delayed truck shipments not only result in increased operational costs but also impact customer satisfaction. Timely delivery of goods is essential to meet customer expectations and maintain the competitiveness of logistics companies.

By accurately predicting truck delays, logistics companies can:

- Improve operational efficiency by allocating resources more effectively
- Enhance customer satisfaction by providing more reliable delivery schedules
- Optimize route planning to reduce delays caused by traffic or adverse weather conditions
- Reduce costs associated with delayed shipments, such as penalties or compensation to customers

In the first project of our three-part series, **Learn to Build an End-to-End Machine Learning Pipeline - Part 1**, we laid the groundwork by utilizing PostgreSQL and MySQL in AWS RDS for data storage, setting up an AWS Sagemaker Notebook, performing data retrieval, conducting exploratory data analysis, and creating feature groups with Hopsworks.

In **Learn to Build an End-to-End Machine Learning Pipeline - Part 2**, we focused on building a machine learning pipeline by retrieving data from the feature store, performing train-validation-test split, one-hot encoding, scaling numerical features, experimenting with logistic regression, random forest, and XGBoost models, and deploying a Streamlit application on AWS.

Building upon the groundwork laid in Parts 1 and 2, this part implements model monitoring to detect data and model drift, integrates CI/CD practices for automated model deployment, and leverages Amazon SageMaker Pipelines for streamlined orchestration of the machine learning workflow. The aim is to ensure the reliability, scalability, and efficiency of the deployed model in real-world logistics scenarios.

Note: AWS Usage Charges

This project leverages the AWS cloud platform to build the end-to-end machine learning pipeline. While using AWS services, it's important to note that certain activities may incur charges. We recommend exploring the AWS Free Tier, which provides limited

access to a wide range of AWS services for 12 months. Please refer to the **AWS Free Tier page** for detailed information, including eligible services and usage limitations.

Aim

The aim of Part 3 is to develop an end-to-end machine learning pipeline by implementing model monitoring, CI/CD practices, and Amazon SageMaker Pipelines for improved reliability and efficiency in logistics operations.

Data Description

The project involves the following data tables:

- City Weather: Weather data for various cities
- Routes: Information about truck routes, including origin, destination, distance, and travel time
- Drivers: Details about truck drivers, including names and experience
- Routes Weather: Weather conditions specific to each route
- Trucks: Information about the trucks used in logistics operations
- Traffic: Traffic-related data
- Truck Schedule: Schedules and timing information for trucks

Tech Stack

- ➔ Language: Python 3.10
- ➔ Libraries: NumPy, Pandas, Evidently AI
- ➔ Data: Hopsworks Feature Store
- ➔ Experiment Tracking: Weights and Biases
- ➔ Model Building: Scikit-learn, XGBoost
- ➔ Cloud Platform: AWS Sagemaker, Amazon SageMaker, AWS Lambda, AWS SNS, AWS CodePipeline, AWS CodeCommit, AWS CodeBuild

Approach

- Setting up Streaming Data Generation:
    - Define functions or scripts to generate streaming data mimicking real-world scenarios
    - Ensure data is representative and diverse to capture various conditions and events
- Lambda Function Creation for Streaming Data:

    - Write Lambda function code to handle incoming streaming data
    - Configure necessary permissions and triggers for Lambda execution
    - Develop Lambda function logic to process and transform streaming data
    - Integrate Lambda functions into the pipeline architecture for seamless data flow
- Docker Image Creation with Python 3.10:
    - Build a Docker image containing the required Python environment and dependencies
- Streaming Data Integration and Pipeline Monitoring:
    - Implement mechanisms for monitoring pipeline, performance, and data quality
- Data Drift and Model Drift Calculation:
    - Implement algorithms to calculate data drift and model drift based on historical and real-time data
    - Monitor deviations in data and model performance for proactive intervention
- Model Retraining and Evaluation:
    - Develop processes for retraining models based on detected drift and updated data
- Scenario-based Runs and Testing:
    - Perform scenario-based runs to simulate different operational conditions and scenarios
- Scheduling SageMaker Pipeline Execution
- Configure scheduling mechanisms to automate the execution of SageMaker pipelines at predefined intervals
- Notification System Setup and Implementation:
    - Configure notification systems to provide alerts and updates on pipeline status and events
    - Integrate notification systems into the pipeline architecture for effective communication and monitoring
- Resource Cleanup Post-execution:
    - Ensure efficient resource management and cost optimization by removing unused or unnecessary resources
- Conclusion:
    - Summarize key findings, insights, and outcomes from the monitoring approach

Code Folder Overview:

Once you unzip the code.zip file, you can find the following folders:

```
├─ Lambda_function_streaming_data
│  ├─ lambda_layers
│  │  ├─ pymysql_layer.zip
│  │  └─ sqlalchemy_psycopg2_layer.zip
│  └─ lambda_streaming_data_gen.py
├─ Python3.10_docker
│  └─ Dockerfile
├─ README.md
├─ sagemaker_pipeline_notifications_cdk
│  └─ python
│     ├─ app.py
│     ├─ cdk.json
│     ├─ Dockerfile
│     ├─ example-workflow.json
│     ├─ lambda_function.py
│     ├─ requirements.txt
│     ├─ resources
│     │  ├─ sample-event.json
│     │  └─ statemachine.png
│     └─ source.bat
├─ StreamingData_Sagemaker_Pipeline
│  ├─ calculate_data_drift.py
│  ├─ calculate_model_drift.py
│  ├─ feature_engg_finalmerge_hopsworks.py
│  ├─ fetch_streaming_dump_to_hopsworks.py
│  ├─ is_first_day_of_week.py
│  ├─ model_drift_not_detected.py
│  ├─ model_retraining.py
│  ├─ not_first_day_of_week.py
│  ├─ pipeline-structure.png
│  ├─ pipeline.py
│  ├─ previous_data_updation.py
│  ├─ sagemaker-pipelines-project.ipynb
│  └─ test.py
├─ Streaming_data
│  ├─ streaming_city_weather.csv
```

```
│   ├── streaming_route_weather.csv
│   ├── streaming_schedule.csv
│   └── streaming_traffic.csv
└── Training_Codes
    ├── app.py
    ├── config.yaml
    ├── data
    │   ├── Database_backup
    │   │   ├── truck-eta-mysql.sql
    │   │   └── truck-eta-postgres.sql
    │   ├── data_description.pdf
    │   └── Training_data
    │       ├── city_weather.csv
    │       ├── drivers_table.csv
    │       ├── routes_table.csv
    │       ├── routes_weather.csv
    │       ├── traffic_table.csv
    │       ├── trucks_table.csv
    │       └── truck_schedule_table.csv
    ├── engine.py
    ├── logs
    │   ├── truck_eta_error_logs.log
    │   └── truck_eta_info_logs.log
    ├── ml_pipeline
    │   ├── data_prep.py
    │   ├── evaluate.py
    │   ├── modelling.py
    │   ├── process.py
    │   └── utils.py
    ├── models
    │   ├── log-truck-model.pkl
    │   ├── randomf-truck-model.pkl
    │   └── xgb-truck-model.pkl
    ├── notebooks
    │   ├── Truck-Delay-Classification_Part_2.ipynb
    │   └── Truck_Delay_Classification_Part_1.ipynb
    ├── output
    │   ├── truck_data_encoder.pkl
    │   └── truck_data_scaler.pkl
    ├── requirements.txt
```

```
└─ sample-monitoring_pipeline.ipynb
```

Here is a brief information on the folders:

- **Lambda Function for Streaming Data**: Contains lambda functions and required layers for streaming data, including lambda_streaming_data_gen.py and zipped Python layers (pymysql_layer.zip and sqlalchemy_psycopg2_layer.zip).

- **Docker Setup**: Provides a Dockerfile for setting up a Python 3.10 environment, located in the Python3.10_docker directory.

- **SageMaker Pipeline Notifications CDK**: Includes scripts and configurations for SageMaker pipeline notifications using CDK, located in the sagemaker_pipeline_notifications_cdk/python directory, with key files like app.py, lambda_function.py, and cdk.json.

- **Streaming Data SageMaker Pipeline**: Contains scripts, Jupyter notebooks, and other resources for the streaming data SageMaker pipeline, found in the StreamingData_Sagemaker_Pipeline directory, with files like pipeline.py, calculate_data_drift.py, and sagemaker-pipelines-project.ipynb.

- **Training Codes**: Includes all the necessary scripts, data, and configurations for training models, located in the `Training_Codes` directory. Key components include training data (`Training_data`), database backups (`Database_backup`), machine learning pipeline scripts (`ml_pipeline`), trained models (`models`), and notebooks for analysis (`notebooks`).

Project Takeaways

1. Gain a comprehensive understanding of the end-to-end machine learning pipeline
2. Learn the importance of modular code development for scalability and maintainability
3. Understand the importance of model monitoring and ensure continuous performance evaluation
4. Learn about various types of drift, including data drift, concept drift, and prediction drift
5. Understand the importance of CI/CD practices and integrate them into the pipeline for automated deployment

6. Explore the benefits and components of AWS SageMaker Pipelines for efficient machine learning workflow management
7. Learn to create and integrate Lambda functions
8. Create Docker image with specific Python versions for consistent environment setup
9. Learn to calculate and address data and model drifts to ensure model reliability using Evidently AI
10. Configure SNS topics to facilitate communication and notifications within the pipeline
11. Develop processes for retraining models based on detected drift and updated data
12. Automate pipeline execution scheduling to ensure timely and consistent performance